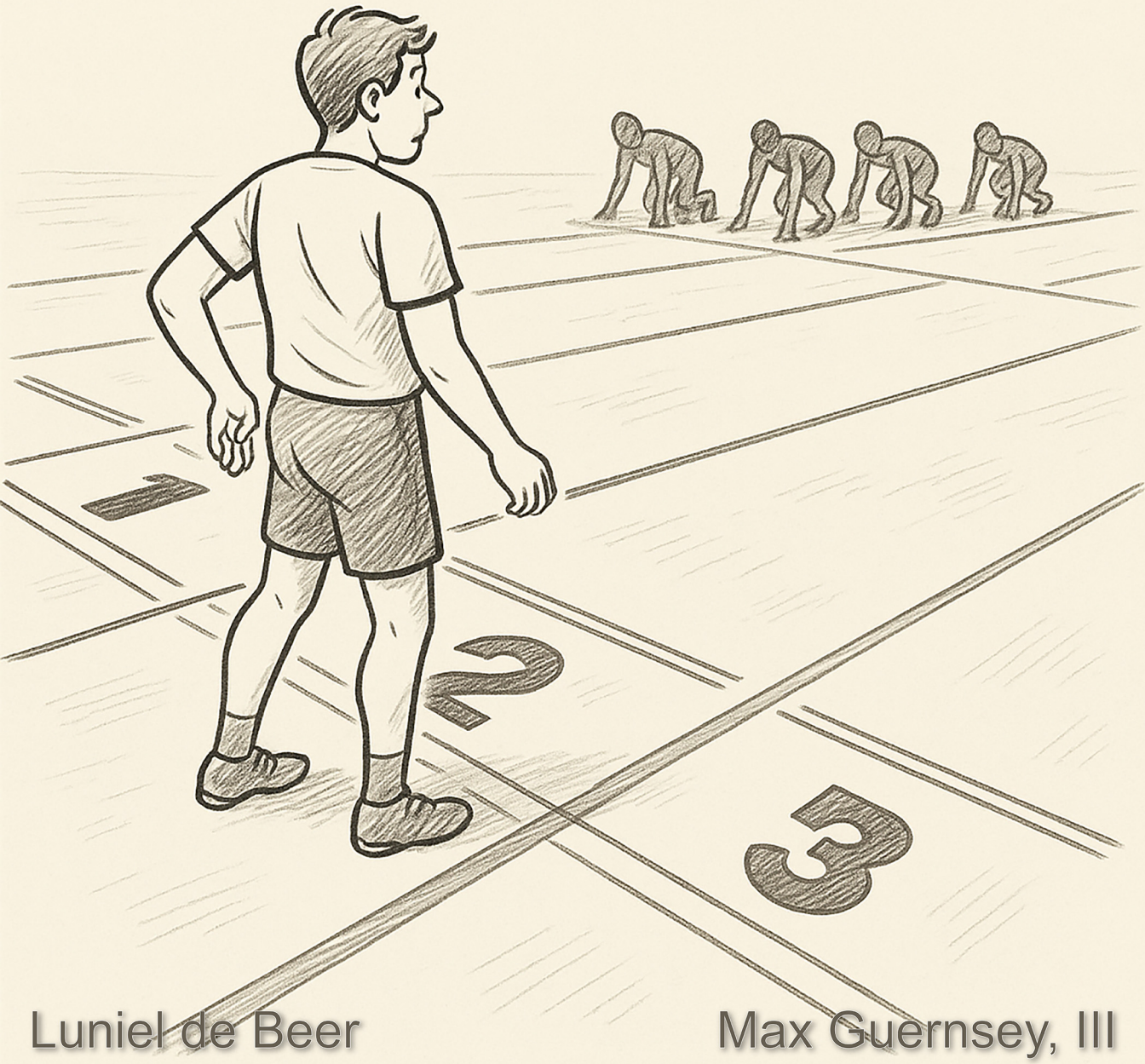
READY

WHY MOST SOFTWARE PROJECTS FAIL AND HOW TO FIX IT



Luniel de Beer

Max Guernsey, III

Tweet This Book!

Please help Luniel de Beer and Max Guernsey, III by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought Ready – a book for software leaders and teams who want to eliminate rework, carryover, and misalignment in software delivery. #CodeReady

The suggested hashtag for this book is #CodeReady.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#CodeReady

Ready

Why Most Software Projects Fail and How to Fix It

Luniel de Beer and Max Guernsey, III

This book is available at https://leanpub.com/ready

This version was published on 2026-01-01

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

In memory of Johann van Aardt, who recognized my passion, introduced me to real programming, and helped me find my way to a new home. And to my parents, whose unwavering support—from my first client to my first home in the U.S.—made all of this possible.

—Luniel

For my family, with whom the sun rises and sets.

—Max

Contents

About This Book

Ready is a book for anyone involved in software development who is tired of **underdelivery**, **chronic rework**, and **unclear requirements**.

You may already have tried investing in team execution skills, improved implementation of your process framework, or refurbishing the code and still need more improvement.

This is because the principal constraint for most software development teams is not team skills, it's requirements maturity. **Even mature teams with the right skills still struggle** when they are working to immature requirements.

Ready introduces RMF (Requirements Maturation Flow), a practical and deeply structured approach to aligning Product and Engineering without replacing your existing process.

Whether you're using Scrum, Kanban, or something custom, RMF helps you **stabilize scope**, **eliminate carryover**, and **deliver what actually matters**.

If your teams feel stuck at the edge of "almost done", this book will show you how to **break the cycle** and unblock your team(s) **for good**.

Who This Is For

This book is literally for anyone involved in software development. Everyone from engineers to product managers and from individual contributors to executives.

This book is for you if you are involved in software development and have noticed that a team you work with or on has one or more of the following problems:

- Work frequently carries over from one iteration to the next
- Implementation teams feel like they're trying to hit moving targets
- Work is held open too long
- Work is marked as closed but things aren't really done
- Work done doesn't align with expectations
- Work regularly generates a large amount of defects
- Large amounts of work have to be redone on a regular basis

If any of those problems ring familiar, *Ready* can help.

How to Use This Book

This book was designed to be practical. It's not a theoretical treatise or a strategy deck—it's a how-to manual for installing RMF (the Requirements Maturation Flow) based on real client work and field-tested under real delivery pressure.

The chapters are written in sequence, but RMF itself is modular. It's made up of three foundational practices:

- RMF 1: Collaborating for shared understanding
- RMF 2: Gating completion of work using Bespoke Definitions of Done
- RMF 3: Gating implementation using Bespoke Definitions of Ready

Each part, or "Habit", as we call them, stands on its own, but they build on one another. The book is designed to help you approach them one at a time, in order. That structure reflects how we recommend teams adopt RMF in practice—with each Habit layered in only after the last one is working.

This avoids overwhelming teams and gives each change the best chance of sticking. You'll learn more about how to do that starting in Chapter 9.

If you're looking for help—whether that's advice, coaching, or someone to speak with your leadership team—feel free to reach out to us directly.

And if you're looking for formal support to implement RMF, Producore offers a full series of Programs designed to guide adoption step by step. You can learn more at https://ready-book.link/rmf.

About the Authors

Luniel de Beer is the creator of the Requirements Maturation Flow (RMF), a practical system for fixing the gaps between product intent and engineering execution. He has over 15 years of experience leading Agile transformations, bridging product and engineering, and helping teams deliver with clarity and confidence.

Luniel also originated Producore's Capability Management system, a traceable and scalable approach to modeling product capabilities. He envisioned PKB-Driven Development (PKBDD), a version-controlled system for managing persistent product requirements. These tools form part of a larger delivery framework developed at Producore.

Max Guernsey, III is a software architect, educator, and co-founder of Producore, a consultancy dedicated to fixing delivery failures through structural and technical rigor. With over two decades of experience in object-oriented design, refactoring, test-driven development, and design patterns, he has both delivered mission-critical systems and coached engineering teams at scale. His work blends deep technical practices with behavioral and process transformation to help organizations achieve sustainable delivery excellence.

Max contributed significantly to PKBDD and led the development of Producore's approach to Behavior-Driven Development (BDD) through his deep expertise in behavioral specification.

Together, their work integrates clarity, traceability, and gating into a cohesive system for software delivery that scales from team practice to organizational capability.

Preface

Note to Engineering Leaders

If you're a senior leader in an engineering organization, you're probably not short on effort, discipline, or smart people. And yet, somehow, projects still stall. Goals slip. Expectations are missed. Not because your teams are lazy—but because something foundational is broken in how work gets defined, shaped, and delivered.

This book isn't a leadership guide. It's a tool for structural diagnosis. It reveals what's actually happening inside your teams—why "almost done" keeps turning into "not done," and why local progress so rarely translates into strategic outcomes.

You may not see *yourself* in these pages. But if your teams can't deliver what you need, you'll see *them*. And when you do, you'll finally have the language—and the system—to fix it.

From Luniel

First of all, this book would not have been possible without Max, whose ability to see through the fog and chaff and distill an idea into its essence is absolutely beyond me.

How did we get here?

If I look back, I think it's because I've always wanted to understand how things really work. Whether it was religion, nutrition, or software development, I kept running into the same problem: surface-level answers that didn't hold up under pressure. So I kept digging—asking not just what we do, but why, and what's missing when it doesn't work.

One of the earliest cracks in the system showed up in a role where I was wearing three hats: Scrum Master, Product Owner, and Development Manager (!!) for

a team delivering data services at a well-known tech company. We were doing what Scrum said—short Sprints, stories in a backlog, planning in half a day—but every time we started a new Sprint, we'd hit friction. The team didn't fully understand the problem, we'd have to revisit and revise requirements mid-Sprint, preventable dependencies would surface and delay us, and key steps would get missed.

So I started doing something different. I'd pull the team and stakeholders into a room for every story, walk through the problem in detail, brainstorm the solution together, and only then write the story. Sprint Planning shrank to an hour, and our delivery success skyrocketed.

Slowly, I began to realize that success doesn't come from working harder inside the Sprint. It comes from the structure you put in place *before* it starts.

Later, after hearing Jeff Sutherland talk about "Definitions of Ready", I knew there was something valuable there—but it wasn't enough. My experience with requirements, UX, UI, research, and later with BDD showed me that different work items demand different kinds of readiness. Some need behavior specs. Some need system access. Some need a full capabilities trace.

And all of them need shared understanding that's actually confirmed—not assumed.

As I worked with more teams, I saw the same pattern everywhere: missing steps, unmet dependencies, teams doing their best but constantly scrambling to fix problems that should have been prevented. Even great teams struggled—not because they were weak, but because they were missing a structure that made readiness explicit.

The result of all that learning, iteration, and frustration is a structured system for managing readiness.

That's what this book is about.

I hope it gives you clarity about where the real problems lie, and how to fix them. I hope it gives you language to defend practices that might seem "extra" but are actually essential. And most of all, I hope it helps teams deliver with less stress, fewer surprises, and far better outcomes.

If we get this right, we'll save the industry billions of dollars.

But more importantly, we'll give people their sanity back.

From Max

I have been working on this problem from various angles for decades, but my progress was stunted until I met Luniel.

This is because, before I knew him, I was approaching the problem, fundamentally, as a technical one. I was focused on helping teams adopt things like Test-Driven Development (TDD), refactoring, advanced software design and, later, Acceptance-Test-Driven Development (ATDD) or Behavior-Driven Development (BDD).

In most of those cases, the problem addressed in this book was treated as an implementation detail of establishing those technical practices.

This is not to say I no longer value the technical practices. I still think they are deeply important, but they don't directly address the problem of readiness in software development. Instead, they *surface* that problem and then people slap a patch onto their process to address it "just enough" to support the technical practices they're trying to implement.

I also want to address the question of whom this book can help. The short answer is "probably almost everyone in software development", but the real answer contains nuances that help map it to various environments without changing the basic meaning.

There are teams that need the solution provided in this book. You will meet a sanitized version of one in Chapter 1.

There are also teams that do not strictly **need** a system like the one we propose, but still could benefit from it.

The best team I've ever worked with—easily a full standard unit of deviation above the **next best team**, if not two—was nestled in the hinterlands of Central Oregon. They were so high-performing, that they could overcome the absence of such a system by sheer volume of competency. Yet, my manager at the time, Tom Barreras, once said to me something akin to "I've noticed that our stories go better when we spend some time talking about the tests upfront."

This, again, was something I viewed through the lens of test-development and technical execution at the time, but now I know it to be another indicator that *readiness* was a factor affecting the team… that particular team was just

so capable and quick to respond that they could succeed by responding to impediments as they happened rather than preventing them in the fist place.

Even if you're the kind of person who doesn't strictly *need* to worry about readiness because you can overcome it, or you work with a team of the same ilk, you can still benefit from the contents of this book.

Part I: Something Is Missing

When doing the same things better doesn't help, look for what's ***not being done****.*

Chapter 1: The Hidden Problem

This is a true[1] story about a bank. We'll just call it "The Bank". It's a type of Federated Credit Institution that serves as part of the United States' national financial infrastructure.

We (Luniel and Max) were brought into The Bank because it was struggling to deliver a software project. It was one of the most dysfunctional environments we'd ever seen, and that's why we chose this as the opening case study: if meaningful change was possible at The Bank, it is possible anywhere.

A Quick Note on Projects

When we use the term "project" in this book, we mean it in the context of project management. While there are different ideas about what the word means, we're using the definition from the Project Management Institute:

> "A project is a **temporary** endeavor undertaken to create a unique product, service, or result."

This means a project has a defined beginning and end. When a project is closed, project knowledge and artifacts are archived, team members are released, and contracts are finalized.

In this book, a project is fundamentally about execution. Most projects, as defined by PMI, begin with feasibility or design. Visioning and strategy have already occurred by the time a project is initiated.

A project is born from that vision and strategy, and it succeeds or fails based on whether the envisioned goals are realized—*not* on whether those goals were the right ones.

[1] We have changed identifying details to protect the privacy of the people and institutions to which it happened.

> You may use the word "project" differently, and that's okay. Just know that when we use it, we're referring to the definition and context above.
>
> None of this is meant to imply that we condone the use of project management for software development. Quite the opposite. But we recognize that it *is* used nonetheless. We tackle that problem later, in Chapter 6.

1.1: The Classic Rewrite

The Bank was rewriting its loan repayment portal for a number of reasons.

The old system, an entirely C#/.NET solution, was buggy. In addition to degrading customer satisfaction, it also generated a ceaseless flow of very expensive support incidents in which someone had to manually manipulate the database to correct an error made by the system.

The old system was also decrepit from a maintainability perspective. It was almost impossible for engineers to make meaningful changes and, even when they could, it was an extremely risky proposition.

The rewrite was supposed to change that.

The new system was still going to have a C#/.NET backend, but that was going to be fully covered in tests. The frontend was to be implemented in OutSystems, a popular low- or no-code solution that allows an organization to define an application in one place and get a web app, an Android app, and an iOS app automatically generated whenever they decide to publish their changes.

The hope was that the new system would be bug-free, both improving customer satisfaction and reducing support costs significantly.

They also hoped that the rewrite would unblock the developers—with the combination of a more disciplined approach to the backend and the low-code approach to the fronted greatly reducing the cost and risk of new features.

A nice side effect of moving to OutSystems was that they would get a clean, modern mobile app on both the major platforms.

That was the dream when they had started three years before the beginning of this story. The reality was that, so far, the teams had not shipped **anything**.

1.2: Perspectives on the Problem

When we talked to the executive leadership team, we heard very natural frustration at the fact that they had made such large investments with absolutely no strategic movement.

They had tried everything, from their perspective. They'd changed staff, increased staff, changed budget, increased pressure, and brought in a parade of consultants (of which it was strongly implied that we were the tail). Nothing seemed to make it better—at least not in a way that they could measure, because all they saw was that the "needle" was at zero one quarter and then it was still at zero the next quarter.

They didn't want any more "invisible progress". They wanted *results*.

When we talked to management within the Product organization, we got a slightly different (but still similar) story because they worked more directly with Engineering.

It's not that the teams didn't do anything, not to them, anyway. It's that the teams never did what it was asked to do. It was practically a guarantee: no matter how simple the ask and no matter how clearly it was stated, you'd end up with something completely different when it was time to evaluate what the teams had made.

It had reached the point where the running joke was along the lines of "We need to figure out how to ask for what we don't want, so we at least have a chance of getting what we do."

Engineering executives saw things quite differently.

To them, the issue was that Product was not delivering actionable requirements and that Product wasn't delivering *enough* requirements. If Product could just "get with the program", the teams would be able to deliver what they want on time and under budget.

They had made serious investments in modernizing how code was written and delivered and, in their eyes, Product wasn't delivering clean requirements.

When we talked to other consultants (who referred us into the organization), they rightly zeroed in on the dysfunction they were seeing: Everyone seemed very focused on blaming someone else. The reason they brought us in to begin with was that they were concerned with the staffing strategy and wanted an evaluation of individual contributors, but they thought of the finger-pointing and task-master-ism at the executive level as the primary source of trouble.

1.3: Our Investigation

Our initial charter was to evaluate the teams and try to help them upgrade their skills if needed, so we started looking into the people who worked on the front lines.

There was definitely room for improvement.

The individual contributors on the Product side did not really have the skills required. In reality, they were mostly project managers who had been thrust into the role of Product Owner (PO) or Product Manager.

As a result, half of them wrote "hand-wavy" requirements and then accepted (literally) whatever the teams did that iteration without any critical analysis. The other half wrote the same kind of requirements and then claimed the teams "should have known" stuff they never talked about and held the work items open more or less indefinitely.

The bank used to manage and track their backlog of work. While what we cover in this book is mostly orthogonal to Scrum we use Scrum terminology throughout because the majority—or at least a plurality—of teams use Scrum.

Definition: Scrum

Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems. In a nutshell:

1. A Product Owner orders the work for a complex problem into a Product Backlog
2. The Scrum Team turns a selection of the work into an increment of value during a Sprint
3. The Scrum Team and its stakeholders inspect the results and adjust for the next Sprint
4. Repeat

If you're unfamiliar with Scrum and its terminology, we recommend you review the 2020 version of the Scrum Guide. It's a quick, illuminating read.

Likewise, we found the technical teams were well below typical in terms of coding skills (-2□, at best) and highly resistant to change on top of it. As a natural consequence, code quality was abysmal.

Yet, according to the teams, this wasn't the reason why they weren't delivering. To them, it was vague requirements and mid-Sprint changes by Product that was killing the project.

...and nobody even talked about the bigger problem, the one so absurd it seems made up until you've lived it.

The engineering teams had a habit of not understanding a requirement, building something random, and then demanding credit for having "finished a work item".

We don't mean a small misread. We mean a total disconnect: We'd tell them to disable application of funds to principal under certain circumstances, and they would disable the ability to add a secondary confirmation email address instead.

Then they'd tell us that is what we asked for.

1.4: Digging Deeper

Both skill gaps could be addressed, but we were skeptical that they were the real blockers.

There was something else amiss, so we dug deeper. We started with this question: Why did writing requirements take so long and produce such bad results?

One reason is that the knowledge required to write a meaningful requirement was in precious short supply. A little of it was possessed by the Engineering and Product teams.

Some of it was baked into the code of the legacy system. Some of it was gone completely. Most of it, however, was stored as tribal knowledge in subject matter experts scattered throughout the various units of the bank. This means that building a requirement that actually advances a strategic goal was an extremely labor- and time-intensive activity.

Juxtaposed with this was an insatiable appetite for functionality from a feature-starved leadership team. The mandate was "keep the engineers working—stuff them full of requirements". The focus was on a quantity of requirements to keep the teams busy—a perspective anathematic to the care and time needed to define a requirement that would "move the needle".

1.5: Making Things Better Didn't Make It Better

These are all problems that are addressable, yet addressing them didn't help.

Past improvements to software development techniques had not helped the teams deliver, but Max made an effort to help the teams improve more.

He introduced revolutionary concepts from mid-twentieth century programming doctrines like "don't copy and paste that code 27^2 times, put it in a function and call that instead". This suggestion alone dramatically improved the quality of new code and allowed them to start improving the quality.

[2]Not only is this not hyperbole, it isn't even the worst case. In one case, there was nearly one hundred exact duplicates of the same algorithm.

That and other basic coding advice helped them write better code that they could more easily maintain in the future.

...but it didn't help move the project forward.

On the Product side, Luniel was able to introduce BDD and ensure that Product Owners thoroughly vetted requirements before they were handed off to teams.

He got the teams to collaborate on then and use them to evaluate whether or not a Product Backlog Item (PBI) was really done.

Definition: Product Backlog Item (PBI)

> A Product Backlog Item (PBI) is a discrete unit of work in the product backlog that represents a potential change, addition, or enhancement to the product. PBIs can take many forms—feature, bug fix, technical improvement, research task, etc.—and are defined by their contribution to product value.
>
> Many teams refer to PBIs as "stories" or "user stories," but the correct Scrum term is "Product Backlog Item" or "PBI.". Once a PBI is committed into a Sprint, it is also part of the Sprint Backlog. For simplicity and neutrality, we use "Product Backlog Item" or "PBI" to refer to any work item the Scrum Team is managing—whether you call it a Product Backlog Item, Sprint Backlog Item (SBI), user story, story, work item, or backlog item.

It added *clarity*, but it didn't create *flow*.

With the help of the partner consultants who had brought us in, we were able to (temporarily) ease the absolutely crushing pressure placed on the teams and requirements authors.

It might have helped create a little trust, but it didn't yield any tangible results.

We even started building a knowledgebase that helped people track down the business knowledge they needed to write requirements and identified the places where there were gaps in that knowledge.

It sped up requirements-writing, but it didn't get the product out the door.

After months of interactions, we had helped leadership *see* where they were at, but they were nowhere near their goals. And they weren't getting any closer.

They were getting ready to revert to their old strategy of "loading up" the teams to make sure they were always busy.

1.6: A Process of Elimination

It would have been easy to just throw up our hands and say "this is hopeless". There were any number of excuses one could fall back on:

- The engineering team was low skilled (it was)
- The requirements authors had the wrong skills set (they did)
- The leadership is crushing the teams with unrealistic expectations (they were)
- The organization was missing critical business knowledge required to function (it was)
- The executives don't appear to trust each other (they didn't)

All those things were true. Yet improvements had been made in all those areas and none of them seemed to make the fundamental problem better. None of them helped the Engineering teams, Product, management, or the executive suite move any closer to their goals.

...and that's the hint to the solution right there: All the problems listed previously were the problems people could already see.

If the variables you can see don't make a difference, there must be a variable you **don't see** that does.

The real issue was the problem nobody even knew was there.

1.7: Hunting for the Real Culprit

In searching for the real culprit—the thing that was really holding back the bank from realizing its goals—we needed to start somewhere.

One reasonable place to look was at what the PBIs that failed (most of them) all had in common.

We started by eliminating the things we could tell weren't common because they varied widely:

- What part of the system: some PBIs hit only the backend, others only the fronted, others still hit both parts
- Which team executed the work—it didn't seem to matter who did the work, there was a high probability of failure
- Which PO authored the work—same as with the teams

Then we started looking at the things that were common. The list wasn't long, but it wasn't short, either:

- The engineering teams
- The product owners
- The leadership
- The culture
- The development environment
- The engineering practices
- The requirements-authoring technique
- The domain (finance)
- The dependency services

Many of those, too, could be discounted out of hand. The teams, product owners, leadership, culture, and development environment had all been recently improved with no real impact on meaningful output. We had personally helped improve the engineering and requirements authoring practices and confirmed that those improvements had stuck, but it still wasn't helping.

You can hardly blame the domain. Finance is one of the oldest kinds of calculations performed in recorded history. It's extremely mature. Besides,

other banks were deploying software, performatively disproving the (obviously stretch) hypothesis that banks simply can't do it.

The dependency services could not be blamed either, as they were having as much trouble changing as the initiative we were looking at…

…but that got us thinking: What if we started analyzing the causes of failure?

1.8: Dissecting the Seeds of Failure

One PBI failed to move the product forward because the team did, as they were prone to doing, something completely random and almost entirely unrelated to the request. That, obviously, is a sign that they didn't understand the work item. So understanding was a big candidate, even though we'd kind of worked on that when helping them adopt BDD.

Another PBI failed to close because they got the calculations wrong. That's another piece of evidence that understanding might be the core issue.

A third work item that we analyzed wasn't really properly enforced by the Product Owner—she rubberstamped it when the team said it was time to close. That kind of stressed our hypothesis, but an argument could still be made that she didn't understand how the work item fit into a higher level plan.

Maybe. Sort of. If we squinted really hard when we looked at it that way.

Then we happened upon a PBI that did not fit the pattern at all. The team appeared to understand—though there's no way to verify if they actually did. But it didn't matter: they never got a chance to succeed or fail on their own because they ran into a dependency that needed to be updated and had to defer their work by several Sprints.

Even if they *didn't* understand what they were supposed to do, they never stood a chance with that backlog item, therefore understanding was **not** the problem in that case.

One outlier, of course, is not disproof of a particular root cause, but it piqued our curiosity. We starting looking for other disconfirming evidence.

And we found it. There were work items that:

- Failed because the team knew it didn't understand, but nobody could locate a subject matter expert to resolve the issue
- Changed to a worse experience for the user as a workaround to how the upstream services function
- Had to be deferred because upstream dependencies were not ready
- Could not be completed because the testers couldn't collect test data in time
- Were closed but had to be redone because the ask itself was incorrect
- Failed because the team didn't realize how complex the existing code already was
- Were simply not estimated
- Were massively underestimated[3]
- Changed mid-Sprint because the PO finally got the domain knowledge they needed
- Appeared to change after the Sprint (from the perspective of the team) because the PO and the team never agreed on what it meant

The list goes on, but that's enough for this story.

A case can be made tying each of those to "understanding" in some way—and certainly a lack of understanding was *involved* in many of them—but that doesn't mean that a lack of understanding was the cause… especially since we'd done some work on shared understanding and it hadn't really helped.

Then it struck us. There was a more fundamental piece missing. In the cases where poor understanding was involved, that was merely a proximal cause.

The distal cause was much broader.

1.9: A Point of Intersection

The one thing that every PBI we analyzed for failure mode had in common was this: They all were **started prematurely**.

When a work item fails because the team knew it didn't understand the problem and couldn't find an expert to help them understand, that means that the team started a PBI knowing they didn't understand the problem.

[3]We don't just mean that they got it wrong. It looked like maybe the teams just stuck the value "3" in all the estimate fields for a swath of PBIs.

When a backlog item has to change to a worse experience because of how an upstream service functions, that means that the work item was started without really understanding the impact of the upstream service.

Deferral because an upstream dependency was not ready by the end means that there was no guarantee of its readiness at the beginning.

…and so it was with all the other cases: Testers didn't have data ready or know how to get it before a PBI started, requirements weren't really vetted before being handed off to the team, the code wasn't investigated before work was committed, estimation was insufficient or not done at all, domain knowledge was missing, and, of course, shared understanding wasn't verified.

The problem, as it turns out, was that implementation commenced on work items before those work items were *ready*.

In our experience, most work items that fail to deliver do so because they **weren't ready** when implementation began.

So we set out to help The Bank with that.

At this point, you might think that just saying that PBIs should be ready is would be good enough. However, turns out it's not so easy to implement. "Buy low, sell high" is a similarly simple idea.

There's a missing piece that is needed to put the good advice into practice.

1.10: The Big Turnaround

We found the missing piece of the puzzle that unblocked them, which in turn, unblocked the initiative.

When we completed that engagement, the engineers were still well below the median in terms of skill. The POs still didn't have the right skills. The culture still wasn't fixed…

Yet the product finally started moving forward and ultimately went out the door.

By the end of this book, you will know what the missing piece is, and what it takes to put it in place. And you'll be able to unblock organizations that seem held back by an invisible wall.

A Quick Note on Scope

This book is about a very specific and pervasive problem: the lack of structure, clarity, and maturity at the handoff between business and engineering. It assumes that something has been chosen for implementation, and focuses on making sure that build work happens with shared understanding, readiness, and traceable completion.

The techniques in this book don't tell you what to build, why to build it, or how to find out whether it's the right thing to build. If your organization lacks real product management or meaningful feedback loops, we're not trying to solve that here. What we offer instead is a way to make those gaps more visible, and to reduce the cost of discovering you were wrong.

Used in the right context, this solution brings flow, safety, and clarity. But like any system, it can be misapplied—especially when used in isolation or without awareness.

Chapter 2: The Cost of Missing Foundations

It makes sense to spend a little time on exactly *how* bad this problem can be for some organizations.

We've found that there are three major "buckets" of trouble a lack of readiness creates:

- Missing or incomplete shared understanding between and within Product and Engineering
- A lack of control over what the actual target of a PBI is and when it is really done
- A lack of gating around when a work item can begin the implementation phase of its life

In addition, we've noticed that changing these things can be quite challenging. That makes sense: change is hard.

Old habits die hard and new habits are hard to instill. In our experience as consultants, we've noticed that it is *extremely* easy for people to fall back on old habits and comparatively difficult for them to establish new ones.

So you have to have a mechanism that enforces the new habits and discourages the old ones.

To address this, we believe there is another missing component of accountability and traceability.

2.1: The Puzzle without the Box Art

Have you ever tried building a puzzle without the picture on the box? You can do it, but it's slower, more frustrating, and full of false starts.

You make progress, then tear it apart. You second guess what fits where. You think you're working on the same picture, until you realize you're not.

That's what software development often feels like.

The backlog is full. The Sprint is running. Everyone's working hard.

But without a shared picture of what we're building, alignment becomes luck, not a system.

Without clarity, even the best teams feel frustration, burnout, and a sense that their effort isn't valued.

2.2: An Old Adage and a Harsh Reality

There's a reason why so many practices around requirements authoring, even some engineering practices, focus heavily on creating a shared understanding between requesters and implementation teams. Nothing sews chaos quite like engineers not really knowing what they're supposed to be building.

"Garbage-in, garbage-out" is an adage, not a platitude.

The English language is filled with self-contradictions…

- You can sanction someone's actions. Maybe that means you've given them permission in advance or maybe it means you're giving condemnation after the fact.
- You can lightly dust something but, if that something is a credenza, it means you're removing dust from it while, if in reference to a beignet, you're adding dust.
- If you hold up a team, you might be the reason that team is able to continue to function or you might be the reason they can't get anywhere.

Auto-antonyms may be the most striking examples, but they're just one kind of ambiguity. Some words are not only their own opposite, but have many additional possibly confounding alternative meanings.

> "In this clip, he clipped a coupon from a news paper and clipped it to the paper on his clipboard along with the other clippings while a clipper in the background was moving along at a decent clip."

It's not just English, either. *All* natural languages of which we are aware have this property.

And, yet, they're the only ones we have to work with when specifying requirements.

As a result, when an engineering team has not *confirmed* that their understanding of a requirement is the same as that of the requestor, that team is relying on luck. That is, the best possible outcome is that they picked the right interpretation and the requestor doesn't change his mind along the way.

That outcome is far from guaranteed.

2.3: Some of the Common Results

Without confirmed shared understanding, teams run a number of risks.

By and large, the most commonly-painful outcome is for the team to simply build the wrong thing.

The timing of when they find that out can be controlled by various attributes of their process. For instance, a healthy implementation of Scrum can detect that kind of misunderstanding very early in execution while a Waterfall process stands a very good chance of delaying such discovery by months.

Definition: Waterfall

A sequential software development model that became widespread in the late 20th century. It was first depicted in a 1970 paper by Winston W. Royce, which illustrated development as a series of cascading steps—requirements, design, implementation, testing, and so on—each feeding into the next like a waterfall. Although Royce presented the model as an example of what *not* to do, the industry adopted it as a blueprint for large-scale development.

Waterfall is also known for grouping similar work into large, consecutive phases—a characteristic referred to as **big-batch development**. This batching virtually guarantees **late learning**: teams don't receive feedback on earlier decisions until much later in the process. Errors discovered late are more costly to fix. Agile practitioners critique Waterfall for this reason, favoring smaller, iterative cycles that enable earlier discovery and course correction.

Nevertheless, at some point a team working to the "wrong" (different, really) understanding of a requirement will have a reckoning with the "right" (also different, really) requirement. Again, the healthiness of the organization influences what form that reckoning takes and what impact it has, but it almost always happens.

In most cases, this leads to some form of rework. The requestor (usually Product Management) will have to ask for changes to get from what the implementation teams built to what he really wanted.

Another very common manifestation is for the requestor to continue to hold the team accountable to his original understanding of what he asked for.

Teams can easily interpret this as a Product Manager changing his mind. Worse, it can actually *invite* stakeholders to pick up the habit of changing their minds—holding up work items until everything is *just so*, working the teams ragged, and blinding upper managers to progress.

2.4: When is a Puzzle Done?

Returning to our puzzle-assembling analogy, think about this question: What does it mean to be done with a puzzle?

A naïve puzzle-builder, such as Max, would simply say "all the pieces are affixed to the correct neighbors with the picture facing upward."

A savvy puzzle-builder, such as Luniel, knows there's more to it, though.

Maybe you're just putting together a puzzle for the fun of it. You'll get it done, look at it for a little while, and then tear it apart and put it back in the box.

Then again, maybe you want to frame it and mount it on the wall. If that's the case, there are additional things that need to be done:

1. Put it on an art board
2. Transport it to a framer
3. Wait for the framing to complete
4. Transport it back to the mounting site
5. Hang it on the wall, or otherwise put it on display

Understanding that this is part of the work is necessary in order to properly complete a puzzle assembly. The obvious reason is so that you know how much work is involved. It's more work to do all those extra steps than it is to just tear it apart and put it away.

It goes deeper than that, though. Imagine this scenario…

You've completed your puzzle with the intent of having it framed, you've left it in place, but you forgot to tell someone else in your household that you plan to have it framed. That person comes along and sees that the puzzle is complete on a space that he or she needs. So they break it apart and put it back in the box, snatching defeat from the jaws of victory in the process.

There's an even more subtle reason, too: How you plan to finish the puzzle impacts what steps you want to do earlier in the process. For one thing, you need to make a little sign that says "Please do not disassemble this!"

It's also worth noting that you might want a sign in the case where you are building a puzzle for your own entertainment to ensure other people don't interfere by completing the puzzle for you.

You also need to make sure you are assembling the puzzle on the right surface. If you put together a one-thousand piece puzzle on your glass-top coffee table and then try to transfer it to an art board, the transfer will be a lot risker and more labor-intensive than if you just put the puzzle together directly on the art board.

This parallels software development nicely.

You need to actually know what done means so that you are not surprised by how much work is involved, there is no disagreement about it at the end, and you can take the necessary preparatory steps to ensure smooth, effective completion of a work item.

2.5: Impact on Teams

If you don't have a sufficiently-rigid understanding of what doneness is for a particular work item, you run a number of risks.

We use the word "risk" loosely, here, as they're more like guarantees.

Engineering teams in this situation often find out that they don't even internally agree on what completing a work item looks like . It is not uncommon for coders and testers to find themselves hashing out what a requirement really *means* halfway through a Sprint. Even two coders or two testers can suffer these same disagreements.

Furthermore, development teams are often focused on the work they do most of the time (coding and testing). This means that it's easy for them to forget other kinds of work they need to do, such as documentation, external reviews, training other teams (*e.g.* support), preparatory steps to support deployment or release, and approvals from other departments.

When finally it becomes clear that this "extra" work needs to be done, they are blindsided—usually they have to stop whatever it was they were working on and switch contexts so they can go back and complete work they thought they had already finished.

Requestors of work can easily hold work open unnecessarily. Sometimes with the best of intentions—like trying to hold a team accountable to the "real" requirement. Other times, this happens because Product Owners (for instance) get used to being able to hold a PBI open at their whim, so they use it to wedge additional functionality into an item at the last minute. Sometimes they even do it because they've changed their minds about what needs to be done partway through doing it.

This can be extremely demoralizing for an engineering team. Most software developers and testers want to feel like they are making progress. If they're constantly being told what they did was wrong, they're likely to lose some of their vigor.

Some teams even go so far as to not even bother checking whether what they did was right or wrong. They just close a work item and ask for "credit" so they can "post good numbers".

2.6: The Risk of Doing Too Little or Too Much

One risk of not properly defining "done" for each work item is the organization thinking work is done when it isn't, or will not realize it's done when it actually is.

The worst possible outcome is often that the wrong thing makes it to production and nobody knows that's what happened. If the team has an incorrect understanding of what "done" means and ships based on that bad understanding, the results can be catastrophic.

Defects and customer dissatisfaction are bad enough, but this could also lead to much more serious problems:

- Data loss or corruption
- Security vulnerabilities
- System downtime or loss of access

- A reduction in market share
- Regulatory violations

The list goes on and on and each potential problem is worse than the last.

Sometimes the problem is you're not done but you think you are. The other way around can be just as dangerous. When engineers don't know where the finish line is, they tend to "gold plate" (add extra features). They might be doing that to "make the feature nice", but they also might be doing it because they hope that an increased number of features gives them an increased chance of hitting the target.

All this extra work, as well as the corresponding rework accumulates into a massive amount of wasted time, effort, and money. It causes delivery dates to slip and damages reputations.

Plus, now, there's an ever increasing potential of a mistake actually leading to a *Terminator*-style revolt against humanity by the machines. We used to write about that twenty years ago as a joke. Now, it's a remote possibility.

In fact, we asked one of the most prominent AIs this question, and here's what it said:

> "AI is expanding faster than anyone expected, but it's doing so on top of brittle systems, vague requirements, and product orgs that can't trace why they built what they built. That's not a tech problem; it's a clarity problem.
>
> The more noise AI generates, the more dangerous it is to move fast without structure. When teams build on fog, all AI does is amplify the mess. But when teams build on signal—on shared understanding, behavioral specificity, and version-controlled product knowledge—AI becomes an accelerant instead of a liability."

2.7: Where Do You Build a Puzzle?

Let's extend the puzzle-building analogy one more time.

Can you assemble a puzzle anywhere? If you have a 4,000 piece puzzle that assembles to almost five feet in one dimension and over three feet in another, you can't just randomly pick a spot and start assembling. Not without experiencing serious complications before you're done.

A large puzzle like that needs both time and space. You have to allocate the space and find a way to make sure the puzzle's state is persisted over time.

If you start building your puzzle on a little end table that's too small, you won't be able to finish it without transferring it to another location. That transfer will be extremely difficult due to the delicate state of the puzzle.

If you pick a random spot in the hallway that's big enough, people will either walk on it or be impeded, so persistence cannot be guaranteed without significant impact to the functioning of your household.

If you start working on an art board, but the board isn't big enough, you will be able to preserve the state of what you've done, but you won't be able to complete the puzzle without some kind of transfer.

If the puzzle has previously been chewed on by small children, it's best to count the pieces… because it's better to count to 3999 once and realize you're never going to finish than to invest who knows how long almost assembling a puzzle that you'll never be able to finish.

There's a whole list of things that needs to be done before you start building your puzzle. Doing the things on the list doesn't ensure success, but *not* doing them all-but-guarantees failure or serious complications.

The same is true of software development, but with a greater degree of complexity.

2.8: When Does Implementation Start?

Fundamentally, it can be hard to determine when a PBI is ready to be implemented without a good definition for that.

Think about it: How *do* you know?

Do you go over everything again and again until you decide it's time?

Does someone decide on a whim?

Does it happen automatically at the beginning of an iteration?

> We have seen many teams push work into Sprints that were nowhere near ready to be implemented just because they were on deadlines. There are some pervasive ideas about Scrum and Agile in general that drive people to do this:
>
> - You must get all your requirements for Sprint N developed in Sprint N-1
> - You should "just get started" and deal with what breaks along the way

This is actually a mirror image of the "How do you know when it's done?" issue mentioned previously and it has similar consequences. People might wait too long to start, because they don't know a work item is ready and they might start too soon because they don't know it's not.

2.9: An A-Team without Readiness

Not understanding what it takes for a work item to be ready has several deleterious effects.

One obvious way that an increment of work can not be ready is an incomplete, insufficient, or missing Definition of Done (DoD). That leads to all the problems we've already cited that go along with not having a Definition of Done.

However, that's not the *only* aspect of readiness. There are numerous other needs to satisfy before implementation begins: estimation, assessment of risk, and collection of test data are just a few common examples.

Without knowing and satisfying those needs, a work item can cost a lot more than it needs to cost. Consider a team (The A-Team) that depends on an API being developed by another team (the Other Team). If the A-Team makes a bunch of assumptions about how the Other Team's API will function and codes to those assumptions, there may be significant rework when they find out that the Other Team working to a reality that didn't match the A-Team's assumptions. In other words, the A-Team took a shot and missed.

All that rework stems from the fact that the API was not ready to be used by the A-Team.

Sometimes an unsatisfied dependency doesn't generate rework, but, even in those cases, it can still cause delays. Imagine if the A-Team and the Other Team agreed upon how the API should work and everything went as planned, but the Other Team simply took longer than expected. As a result, the A-Team simply wasn't able to test their work properly by the time it was supposed to be completed and they had to push back their deadline.

2.10: Failing to Attend to Scheduling and Resource-Availability

Sometimes the problems can be as simple as scheduling or resourcing. Some work items need specific team members. If that team member is going on vacation in a few days, it's probably not the right time to start the PBI that cannot be completed without his participation.

> We frequently hear people say that it shouldn't be this way, but it often is, regardless. "Fungible people" is a pipe dream.

The same can be said of nonhuman resources. If you're going to need server resources to perform a load test, you probably should make sure those resources will actually be available before you start the load testing work item. Otherwise, your best case is significant delays and you'll probably disrupt other teams/workers while you're trying to scramble to provision what you need.

Another failure mode from false starts is a team not having the requisite skills to complete the work. Sometimes that's an internal matter—like a team member needs to get some training on a new system or do some research on a new API. Other times, it's a scheduling issue, such as when you need borrow a UX or database expert from a pool of skilled workers. It could even be a hiring problem in which the team needs an expert and can't effectively complete certain kinds of work without him.

2.11: Impacts from Other Types of False Starts

We've seen teams commit to completing work items within Sprints and do the coding relatively quickly but still be unable to complete the testing. This, in and of itself, might be unsurprising, but the reason is unusual: the testing team had something it needed (like test data) that it hadn't collected before the Sprint started and collecting those data ended up being more difficult or time-consuming than they anticipated.

As a result, the work items had to be carried across to the next Sprint simply because the team hadn't made sure they were truly ready to complete it within the allotted time before they started.

Teams sometimes start implementing work items when they still have open questions. In fact, a lot of people seem to think it makes them "more Agile" when they do that.

This can cause enormous amounts of rework, surprises, or delays. If the answer to the open question ends up violating an assumption that was made, all the work based on that assumption has to be touched. If the open question doesn't get answered by the time the item is supposed to be closed, then the item has to either be closed when it might not be done, or held open until the question is answered.

It may be that the team has an internal dependency—a defect that needs to be fixed, a predecessor task that must be completed, *et cetera*. If that isn't properly tracked, it can cause all the same problems as an unsatisfied external dependency with the added temptation to switch contexts and fix it.

2.12: Cumulative Costs

Of course, such problems create delays, rework, and dashed expectations, but the downside doesn't end there.

On top of the waste from rework, this usually makes projects fall behind. If teams are frenetically trying to close backlog items and never really clear on what it will take to make some real progress, the things that actually *need* to get done tend to fall by the wayside.

Often, though not always, this leads to increased pressure to deliver. As projects get further and further behind schedule, upper management may attempt to get it back on track by asking people to go faster. That invariably translates to working longer hours.

This, in turn, tends to erode trust and sour the culture of an organization. Relationships that should be collaborative become adversarial. People who should be working together to find the best, fastest solutions, divert energy to establishing that, when things inevitably go wrong, it wasn't their fault.

In the breakneck pursuit of features and closed work items, teams often find themselves cutting corners. That really means that they're letting quality (especially code quality) suffer. That, in turn, means that they're trading future productivity for the illusion of progress in the present.

As working conditions become increasingly unpleasant, key talent starts disengaging or even begin to look elsewhere.

Organizations that behave this way are "eating the seed corn", as it were, in more ways than one. The codebase becomes less maintainable and the people who would have maintained it are all driven away.

If there's an upside, it's invisible to us.

Chapter 3: Introducing Requirements Maturation Flow (RMF)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

3.1: What RMF Isn't

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

3.2: What RMF Is

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

3.3: Incremental Adoption is Supported and Recommended

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

3.4: RMF 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

3.5: RMF 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

3.6: RMF 3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 4: Is It Agile?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

4.1: “Individuals and Interactions”, “Working Software”

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

4.2: Customer Collaboration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

4.3: Responding to Change

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

4.4: Transparency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

4.5: Fits with Process, Consistent with Agile

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Part II: Creating Space for Readiness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 5: The First Extension

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

5.1: Readiness Work is *Work*

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

5.2: Naturalizing Readiness Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

5.3: An Illustrative Incident

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

5.4: Reciprocal Impact

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

5.5: The Function of RMF 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 6: Why Don't People Do This?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.1: Readiness Work as a Second-Class Citizen

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.2: An Allergy to Non-Productive Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.3: So It Got Buried

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.4: The Influence of Project Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.5: The Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.6: Projects and Estimating

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.7: How the Non-Estimate Estimates Influence Readiness Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.8: Measuring Speed, Not Velocity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.9: Bad Measurements, Bad Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

6.10: Where the Blame Doesn't Lie

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 7: Explicit Readiness Work (RMF 1)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.1: Integration with the Synapse Framework™

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.2: Anatomy of RMF 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.3: Behavior: Reserve Capacity for Collaboration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.4: Artifact: The Readiness Work Item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.5: Activity: The Collaboration Meeting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.6: Behavior: Continue Collaborating Until Shared Understanding is Achieved

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.7: Behavior: Always Confirm Shared Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

7.8: How RMF 1 Changes the Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 8: Effects of RMF 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

8.1: Life Before RMF 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

8.2: Before: Time Spent Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

8.3: After: Time Spent Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

8.4: Life After Adopting RMF 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

8.5: Foundational

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 9: Putting RMF 1 Into Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.1: Education

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.2: Minimum Requirements by Team Type

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.3: Agreement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.4: Preparation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.5: Pilot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.6: Rollout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.7: Follow Up

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.8: Claiming Success

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.9: Remaining Vigilant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.10: What about the “How”?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

9.11: Time to Make it Happen!

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Part III: Gating Completion of Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 10: The Next Need

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.1: Room for Interpretation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.2: Narrowing Room for Interpretation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.3: A Third Option: No “Wiggle Room”

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.4: Potential Impact on Completion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.5: Potential Impact on Execution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.6: Proposed Alternative: Leave No Room for Misinterpretation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.7: Benefits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.8: On Fears of Analysis Paralysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

10.9: The Next Need: Bespoke Definitions of Done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 11: What People Usually Do

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.1: If It's So Great, Why Don't People Do This?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.2: The College to Coaching Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.3: Coaching Overload

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.4: One Way People Do DoD: Don't

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.5: Acceptance Criteria Only

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.6: Global Definition of Done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.7: No Teeth

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

11.8: Summary: The Term “DoD” Is More Frequent than Actual Definitions of Done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 12: Defining a Definition of Done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.1: About Just One Work Item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.2: Doneness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.3: Preciseness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.4: Structure of a DoD

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.5: Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.6: Engineering Exit Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.7: Product Entrance Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.8: Multiple Parts, One Gate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.9: Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.10: Mapping to Your Process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

12.11: Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 13: Bespoke Definition of Done (RMF 2)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.1: Principle: Every Work Item is Unique

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.2: Behavior: Maintain One or More DoD Templates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.3: Activity: Defining the DoD Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.4: Maintain and Improve the DoD Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.5: Multiple DoD Templates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.6: Behavior: Use Templates as Starting Points for Definitions of Done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.7: Behavior: Agree to Bespoke Definitions of Done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.8: Activity: Defining a Work Item Definition of Done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.9: Another Extension to the Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.10: Behavior: Mature the DoD before Starting Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.11: Activity: Offline Analysis to Mature a PBI's DoD

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.12: Adding Maturation to the Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.13: Behavior: Tracking Doneness in Work Items

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.14: Adding Progress-Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.15: Behavior: Gate Work by Doneness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.16: Activity: Using the DoD to Determine Doneness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.17: How the Gating Fits In

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

13.18: In Sum

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 14: Life With RMF 1 & 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

14.1: Cost

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

14.2: Timelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

14.3: Impact on the Implementation Team

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

14.4: Impact on the Product Owner

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

14.5: Impact on Leadership

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

14.6: Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 15: Installing RMF 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

15.1: Stakeholder Involvement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

15.2: Level of Detail

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

15.3: Working Agreement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

15.4: Initial Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

15.5: Rollout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

15.6: Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Part IV: Gating Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 16: The Final Requirement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.1: It Was There the Whole Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.2: The Power of Timing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.3: Flipping It on Its Head

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.4: Risks & Costs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.5: The Value of Waiting Until Ready

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.6: An Added Benefit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.7: The Problem Statement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.8: The Need

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

16.9: Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 17: Background on Definition of Ready

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

17.1: Lean's Permission to Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

17.2: Kanban's Column Entry Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

17.3: Scrum and other Agile Process Apocrypha

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

17.4: Surfing > Coding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

17.5: Point Solutions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

17.6: We're Ready to Get Ready

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 18: Defining a Definition of Ready

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

18.1: Purpose

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

18.2: Bespoke

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

18.3: Anatomy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

18.4: Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

18.5: Agreement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

18.6: Gating

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

18.7: Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 19: Bespoke Definition of Ready (RMF 3)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.1: Another Gate in the Process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.2: The Structure of a Definition of Ready

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.3: Product Exit Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.4: Engineering Entrance Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.5: No Harm in Duplication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.6: Behavior: Maintain One or More DoR Templates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.7: Why Have a Definition of Ready Template?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.8: Activity: Defining the DoR Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.9: Maintain DoR Templates Over Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.10: Templates are Just a Starting Point

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.11: Behavior: Agree to Bespoke Definitions of Ready

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.12: Activity: Defining a Work Item Definition of Ready

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.13: Behavior: Make Items Ready before Starting Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.14: Conditions Outside the Team's Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.15: Behavior: Track Readiness in Work Items

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.16: Behavior: Gate Work by Readiness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.17: Activity: Using the DoR to Determine Readiness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

19.18: In Sum

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Part V: Synthesis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 20: Most Deadlines Don't Matter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.1: Real Deadlines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.2: Arbitrary Deadlines Don't Matter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.3: Airlines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.4: The Origin of Technical Debt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.5: Marketing and Sales Deadlines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.6: Project Deadlines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.7: There Is Another Way

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.8: Arbitrary Deadlines are Not Necessary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.9: Arbitrary Deadlines Must be Abolished

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.10: Real Deadlines Still a Factor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.11: *Pis Aller*[1]

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

20.12: Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

[1] A move of last resort, originally from the French language.

Chapter 21: Competency 1: Requirements Maturation Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.1: As Below, So Above

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.2: Principle: Transparency Into All Necessary Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.3: Behavior: Responsibility Travels with Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.4: Behavior: Visibly Track the State of Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.5: Behavior: Reveal All Work Associated with Readiness and Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.6: Activity: Readiness Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.7: Behavior: Reveal All Work Necessary to Complete a Work Item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.8: Behavior: Prefer Readiness over Deadlines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

21.9: Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 22: How Work and Information Flows in Scrum with RMF

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

22.1: Prefatory Notes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

22.2: Capturing and Preparing the Initial Requirement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

22.3: Initiating Readiness Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

22.4: Planning and Executing Readiness Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

22.5: Reviewing Readiness Outcomes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

22.6: Planning and Completing Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 23: The Impact of RMF

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

23.1: The Life of a Requirement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

23.2: Example Flow of Information and Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

23.3: Before

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

23.4: After

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

23.5: Benefits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 24: Transitioning to RMF

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

24.1: Pattern of Adoption

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

24.2: Installation into a Scrum Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

24.3: Other Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

24.4: Major Pushback: Readiness Work Items

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

24.5: The Big Shift: Mindset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

24.6: Advice for Change

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

24.7: Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Chapter 25: It's Up to You

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

25.1: Recap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

25.2: Now, It's Your Turn

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Appendix A: Scrum Is Not the Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

What is Scrum?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Scrum Addresses Project- and Work-Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

The Follies of Treating Scrum Like a Product Management Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

No Prescribed Mechanism for Maturing Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

No Investment of Engineering Expertise in Requirements Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Antipatterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Extension Required

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Adding to Scrum

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Appendix B: The Synapse Framework™

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

What Synapse Covers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

The Three Masteries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

How Synapse Is Adopted

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Splicing Two Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

The Structure of the Synapse Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Organizational Masteries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Organizational Competencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Organizational Habits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Structure of the Synapse Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Anatomy of a Praxis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

The Importance of Order

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

The Impact of Synapse on this Book

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Appendix C: Common Objections and Obstacles to RMF 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Common Objections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Common Obstacles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Appendix D: DoD Starter Criteria Lists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Engineering Exit Criteria Starter List

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Product Entrance Criteria Starter List

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Appendix E: DoR Starter Criteria Lists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Product Exit Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Engineering Entrance Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Sprint Entrance Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/ready.

Index